UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/754,785	01/04/2001	Pierre-Alain Darlet	40101/06901	3238

30636 7590 09/15/2009

FAY KAPLUN & MARCIN, LLP
150 BROADWAY, SUITE 702
NEW YORK, NY 10038

EXAMINER	
KISS, ERIC B	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
09/15/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 09/754,785
Filing Date: January 04, 2001
Appellant(s): DARLET, PIERRE-ALAIN

Michael J. Macin (Reg. No. 48,192)
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed July 9, 2009, appealing from the Office action mailed December 31, 2008.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

Although appellant asserts that none exist, (Br. 2), the following are the related appeals, interferences, and judicial proceedings known to the examiner which may be related to, directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal:

Appeal 2007-0224, before the Board of Patent Appeals and Interferences, decided May 23, 2007, is directly related to the pending appeal because it was a previous appeal in the same application, and the Board in affirming the examiner's rejections made several findings of fact regarding the *Levine* reference and claim construction that are relevant to the rejections at issue in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

No amendment after final has been filed.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is substantially correct. However, the brief contains multiple references to "p. 32, ll. 7-15," of appellant's specification, where appellant's specification actually ends at page 32, line 8. (Br. 2-3.) Because the disclosure on page 32 of appellant's specification does not appear to directly relate to any claimed features of appellant's invention, the examiner presumes that references to page 32 should simply be disregarded.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

Glen Overby, "Upgrading Your Minix System," 1990 [online], accessed 01/12/2006, Retrieved from Internet <URL: http://www.funet.fi/pub/minix/unsorted/upgrading.txt>, 10 pages.

John Levine, "Linkers and Loaders, chapter 6," June 1999 [online] accessed 08/15/2005, Retrieved from Internet <URL: http://www.iecc.com/linker/linker06.txt>, 9 pages.

6,185,733 BRESLAU et al. 2-2001

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

A. Claims 1-15, 40, 41, and 43-60 are rejected under 35 U.S.C. 103(a) as being unpatentable over John Levine, "Linkers and Loaders, chapter 6," June 1999 [online] accessed 08/15/2005, Retrieved from Internet <URL: http://www.iecc.com/linker/linker06.txt>, 9 pages (hereinafter *Levine*).

As per claim 1, *Levine* discloses receiving a software module, the software module including references to locations within the software module, at least some of the references being backward references; and reordering components of the software module into a predetermined order based on a type (*i.e.* objects being referenced) of the components to remove at least some of the backward references (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper

dependency order to allow a single sequential linker pass to resolve all undefined references), wherein the components include at least one of a header, a section, and a table (see p. 2 (Libraries consist of an archive header, followed by alternating file headers and object files)).

Levine further discloses that under certain circumstances, lorder and tsort won't be able to come up with a total order for the files, resulting in some backward references remaining (see "Exercises" on p. 8). *Levine* further suggests a solution to remaining backward references, which includes listing the same library several times on the linker command line (see "Searching libraries" on p. 6-7), essentially replacing a backward reference with a forward reference to the repeated library entry, thereby loading (storing in memory) the components in such a manner as to avoid a nonsequential reading of the reordered software module, *i.e.*, the components are still read sequentially, but in a longer sequence that includes some repeats (for example, A B A, B A B, or A B C D A B C D, as described on p. 6 of *Levine*). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to utilize such storing of backward references in memory to avoid a nonsequential reading as a known means of handling a known limitation of attempting to remove such backward references.

As per claim 2, *Levine* further discloses adjusting at least one of the references in the software module to reflect the reordering of the components of the software module, so that the at least one of the references remains a reference to the same component, by to the component's new, reordered location, the new, reordered location coming after the at least one reference in the software module (see "Creating libraries" on pp. 5-6 and "Library formats" on pp. 1-5).

As per claims 3 and 4, *Levine* further discloses the software module including a symbol table, the symbol table including no backward references after the reordering and adjusting steps

(see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references). Further, *Levine* discloses that under certain circumstances, lorder and tsort won't be able to come up with a total order for the files, resulting in backward references remaining (see "Exercises" on p. 8).

As per claims 5-8, *Levine* further discloses the use of relocatable ELF object files, which include sections grouped into segments (see "Library formats" on pp. 1-5). *Levine* further discloses the software module including a symbol table, the symbol table including no backward references after the reordering and adjusting steps (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references). Further, *Levine* discloses that under certain circumstances, lorder and tsort won't be able to come up with a total order for the files, resulting in backward references remaining (see "Exercises" on p. 8).

As per claim 9, *Levine* discloses a reorder module configured to receive a software module including references to locations within the software module, at least some of the references being backward references, the reorder module configured to reorder components of the software module into a predetermined order based on a type of the components (*i.e.* objects being referenced) and remove at least some of the backward references (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references), the components including at least one of a header, a section, and a

table (see p. 2 (Libraries consist of an archive header, followed by alternating file headers and object files)). The use of a processor and memory is inherent in realizing the functionality of *Levine*.

Levine further discloses that under certain circumstances, lorder and tsort won't be able to come up with a total order for the files, resulting in some backward references remaining (see "Exercises" on p. 8). *Levine* further suggests a solution to remaining backward references, which includes listing the same library several times on the linker command line (see "Searching libraries" on p. 6-7), essentially replacing a backward reference with a forward reference to the repeated library entry, thereby loading (storing in memory) the components in such a manner as to avoid a nonsequential reading of the reordered software module, *i.e.*, the components are still read sequentially, but in a longer sequence that includes some repeats (for example, A B A, B A B, or A B C D A B C D, as described on p. 6 of *Levine*). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to utilize such storing of backward references in memory to avoid a nonsequential reading as a known means of handling a known limitation of attempting to remove such backward references.

As per claims 10, *Levine* further discloses adjusting at least one of the references in the software module to reflect the reordering of the components of the software module, so that the at least one of the references remains a reference to the same component, by to the component's new, reordered location, the new, reordered location coming after the at least one reference in the software module (see "Creating libraries" on pp. 5-6 and "Library formats" on pp. 1-5).

As per claims 11 and 12, *Levine* further discloses the software module including a symbol table, the symbol table including no backward references after the reordering and

adjusting steps (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references). Further, *Levine* discloses that under certain circumstances, lorder and tsort won't be able to come up with a total order for the files, resulting in backward references remaining (see "Exercises" on p. 8).

As per claims 13-15, *Levine* further discloses the use of relocatable ELF object files, which include sections grouped into segments (see "Library formats" on pp. 1-5). *Levine* further discloses the software module including a symbol table, the symbol table including no backward references after the reordering and adjusting steps (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references). Further, *Levine* discloses that under certain circumstances, lorder and tsort won't be able to come up with a total order for the files, resulting in backward references remaining (see "Exercises" on p. 8).

As per claims 40 and 41, *Levine* further discloses linking the reordered module after the reordering (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references).

As per claims 43-46, *Levine* further discloses the use of relocatable ELF object files, which include sections grouped into segments (see "Library formats" on pp. 1-5).

As per claims 47-54, *Levine* further discloses the reference pointing to/into a section or module before and after reordering (see "Creating libraries" on pp. 5-6 and "Library formats" on pp. 1-5).

As per claim 55, *Levine* discloses receiving a software module, the software module including components arranged in a first order, a first one of the components including a reference to a location in a second one of the components, the second one of the components preceding the first one of the components in the first order; and arranging the components into a predetermined second order so that the second one of the components is subsequent to the first one of the components in the second order, wherein the arrangement is based on a type (*i.e.* objects being referenced) of the first and second ones of the components (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references), wherein the components include at least one of a header, a section, and a table (see p. 2 (Libraries consist of an archive header, followed by alternating file headers and object files)).

Levine further discloses that under certain circumstances, lorder and tsort won't be able to come up with a total order for the files, resulting in some backward references remaining (see "Exercises" on p. 8). *Levine* further suggests a solution to remaining backward references, which includes listing the same library several times on the linker command line (see "Searching libraries" on p. 6-7), essentially replacing a backward reference with a forward reference to the repeated library entry, thereby loading (storing in memory) the components in such a manner as to avoid a nonsequential reading of the reordered software module, *i.e.*, the components are still

read sequentially, but in a longer sequence that includes some repeats (for example, A B A, B A B, or A B C D A B C D, as described on p. 6 of *Levine*). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to utilize such storing of backward references in memory to avoid a nonsequential reading as a known means of handling a known limitation of attempting to remove such backward references.

As per claims 56 and 57, *Levine* further discloses linking the reordered module after the reordering (see "Creating libraries" on pp. 5-6, and in particular, the discussion of using tsort and lorder to arrange object files within an archive library in proper dependency order to allow a single sequential linker pass to resolve all undefined references).

As per claim 58-60, *Levine* further discloses the use of relocatable ELF object files, which include sections grouped into segments (see "Library formats" on pp. 1-5).

B. Claim 42 is rejected under 35 U.S.C. 103(a) as being unpatentable over *Levine* as applied to claim 1 above, and further in view of U.S. Patent No. 6,185,733 to Breslau et al.

As per claim 42, *Levine* discloses such a method but fails to expressly disclose transferring the reordered module to a different computer system and linking the module on the different computer system. However, *Breslau et al.* teaches the use of remote object libraries distributed prior to linking (see, for example, col. 4, lines 11-20). Therefore, it would have been obvious to one of ordinary skill in the computer art at the time the invention was made to such use of a different computer for linking. One would be motivated to do so, for example, to facilitate distributed software development efforts or reduce the physical storage requirements for object files (see, for example, col. 2, lines 4-25).

(10) Response to Argument

A. The rejection of claims 1-15, 40, 41, and 43-60 under 35 U.S.C. § 103(a), (Br. 4-6).

Claim 1, (Br. 4-5)

Appellant argues that *Levine* fails to teach "reorder[ing] components of the software module into a predetermined order based on a type of the components to remove at least some of the backwards references," as recited in independent claim 1. Specifically, appellant argues that *Levine* fails to teach a "predetermined order" and "based on a type of the components." (Br. 5.) The examiner disagrees.

***Levine* teaches a predetermined order.**

As disclosed in the Levine reference,

> *Lorder* **took as its input a set of object files** (not libraries), and produced a dependency list of what files refer[r]ed to symbols in what other files. . . . *Tsort* did a topological sort on the output of *lorder*, **producing a sorted list of files so each symbol is defined after all the references to it**, allowing a single sequential pass over the files to resolve all undefined references. The **output of *lorder* was used to control *ar***.

Levine at p. 5 (emphasis added). The purpose of the topological sort disclosed by *Levine* is to sort the input files such that there are no backward references (symbols defined before references to them), and this sorted list is processed by *ar* to store the reordered object code. *See Id.* The output of the *lorder* command is a "predetermined order", *i.e.*, an ordered list of files with no backwards references, and this predetermined order is issued to the *ar* command to produce the actual reordered output.

In the previous appeal, the examiner cited *Overby* as further illustrating the inherent characteristics of the *lorder* and *tsort* commands in creating a library order, as taught by *Levine*. (Examiner's Answer, May 31, 2006, p. 6.) Specifically, the Overby reference discloses,

> Two utilities are required to create a library order: lorder and tsort. Lorder creates a dependency list, that is, a list of what functions are required by what other functions. Tsort takes the output of lorder and does a "topological sort" **to create an ordering with no backwards references**.

Overby at p. 4 (emphasis added).

Because this ordering with no backward references must be determined prior to actually creating a reordered library, the examiner finds no reason why it cannot be reasonably characterized as "predetermined."

***Levine* teaches a reordering based on a type of the components.**

Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. *See In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

Appellant appears to contend that, "based on a type of the components," must be interpreted as meaning "based on the functions of the various components," (Br. 4-5 (emphasis added) (citing Specification, pp. 5-17, Figs. 1-4).) Despite appellant's contrary assertion, claim 1 does not recite the reordering being, "based on the functions of various components." (*See* Br. 5; Claim 1.)

Where an explicit definition is provided by the applicant for a term, that definition will control interpretation of the term as it is used in the claim. *Toro Co. v. White Consolidated Industries Inc.*, 199 F.3d 1295, 1301, 53 USPQ2d 1065, 1069 (Fed. Cir. 1999) (meaning of

words used in a claim is not construed in a "lexicographic vacuum, but in the context of the specification and drawings"). However, in this case, appellant's specification does not set forth a definition of *type* "with reasonable clarity, deliberateness, and precision" that would render the incorporation of such a definition into the claims appropriate. *See In re Paulsen*, 30 F.3d 1475, 1480, 31 USPQ2d 1671, 1674 (Fed. Cir. 1994). Accordingly, the limitations of claim 1 cannot be properly construed to require the reordering being based on the functions of the various components. Instead, throughout appellant's specification, the word *type* appears to be used to describe merely generic categorization of information (consistent with its ordinary meaning) rather than specifically denoting function (or some other specialized technical meaning). (*See, e.g.*, Specification at p. 6, line 27 ("The software module may have a number of components, including . . . sections of various types."); p. 8, line 23 ("[T]he program header table may be modified to include additional segments or additional types of information."); p. 11, line 11 ("two types of symbol relocation information table entries"); p. 11, line 31 ("two types of backward references").) Further, the word *type* is also used in appellant's specification to describe the categorical method of referencing components. (*E.g.*, *Id.* at p. 9, line 30-33 ("The reference type may indicate whether the symbol is defined in the software module or outside the software module. The reference type may also indicate the way the symbol is used, e.g., direct reference, indirect reference, offset reference, etc."); p. 24, lines 18-29 (reference type); p. 26, lines 14-16 ("type of reference being made to the symbol, e.g., direct, indirect, offset reference, etc.").)

Because the reordering taught by *Levine* is based on a categorization of components, *i.e.*, referenced components, wherein the particular order is based on *how* the components are

referenced, *i.e.*, what symbols are referenced in other files, Levine at p. 5, the reordering may be reasonably interpreted as being based on a type of the components.

Claim 9, (Br. 5-6)

Appellant has merely restated the limitations recited in independent claim 9 and further asserts that the rejection of claim 9 should be reversed, "for the reasons discussed above with reference to claim 1." (Br. 5.) Appellant has not alleged that claim 9 should be considered patentable for any reason separate from the reasons given for claim 1. A statement which merely points out what a claim recites will not be considered an argument for separate patentability of the claim. 37 CFR 41.37(c)(vii). Accordingly, the rejection of claim 9, and claims 10-15 dependent therefrom, should be affirmed for the reasons given above with respect to claim 1.

Claim 55, (Br. 6)

Appellant has merely restated the limitations recited in independent claim 55 and further asserts that the rejection of claim 55 should be reversed, "for the reasons discussed above with reference to claim 1." (Br. 6.) Appellant has not alleged that claim 55 should be considered patentable for any reason separate from the reasons given for claim 1. A statement which merely points out what a claim recites will not be considered an argument for separate patentability of the claim. 37 CFR 41.37(c)(vii). Accordingly, the rejection of claim 55, and claims 56-60 dependent therefrom, should be affirmed for the reasons given above with respect to claim 1.

B. The rejection of claim 42 under 35 U.S.C. § 103(a), (Br. 6-7).

Regarding claim 42, appellant merely argues that the Breslau patent allegedly fails to cure the defects of the Levine reference with respect to the limitations recited in parent claim 1. (Br. 6-7.) Appellant does not argue any error in the applied teachings of the Breslau patent as providing, "transferring the reordered module to a different computer system and linking the module on the different computer system." (Final Rejection 9.) Accordingly, the rejection of claim 42 should be affirmed for the reasons given above with respect to claim 1.

(11) Related Proceeding(s) Appendix

Copies of the court or Board decision(s) identified in the Related Appeals and Interferences section of this examiner's answer are provided herein.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

/Eric B. Kiss/
Eric B. Kiss
Primary Examiner, Art Unit 2192

Conferees:

/Tuan Q. Dam/
Tuan Q. Dam
Supervisory Patent Examiner, Art Unit 2192

/Lewis A. Bullock, Jr./
Supervisory Patent Examiner, Art Unit 2193